Frontier Technical Challenge

Notes

Be sure to write comments, a README and any assumptions. Provide instructions on how to run the project and any notes about your solutions.

Our first choice would be for you to use C++ for this challenge, however if you are unable, feel free to use any of the following languages: JavaScript, Typescript, Python or C# and any framework you see appropriate.

UI is great, but we are most interested to see your problem-solving skills and approach. Keep it simple, keep it DRY and don't overcomplicate or over engineer, comment and test as much as possible.

Please do not use ChatGPT to design any of the solution.

Commit your code to a public Git repository and provide us the URL.

Brief

You have a toy robot on a tabletop, a grid of 5 x 5 units, there are no obstructions. You can issue commands to your robot allowing it to roam around the tabletop. But be careful, don't let it fall off!

Create an app that allows commands to be issued to the robot. The robot should be prevented from failing off the tabletop to its destruction.

A failed command should not stop the app, valid commands should be allowed.

The application should discard all commands until a valid place() command has been executed.

0, 0 on the grid should be seen as bottom left.

Input

Every command should provide visual output that the command has either succeeded or failed.

place(x, y, facing)

x and y are integers that relate to a location on the grid. Values that are outside the boundary of the grid should not be allowed.

facing is a string referencing the direction the robot is facing. Values NORTH, SOUTH, EAST or WEST are allowed.

move()

Moves the robot 1 grid unit in the direction it is facing unless that movement will cause the robot to fall off the grid.

left()

Rotate the robot 90° anticlockwise/counter clockwise.

right()

Rotate the robot 90° clockwise.

report()

Outputs the robot's current grid location and facing direction.

Output

place(0, 0, 'NORTH')

move()

report() => Output: 0, 1, NORTH

place(0, 0, NORTH)

left()

report() => Output: 0, 0, WEST

place(1, 2, EAST)

move()

move()

left()

move()

report() => Output: 3, 3, NORTH